Lógica Proposicional, Álgebra de Boole e Circuitos Lógicos

Nelma Moreira

Departamento de Ciência de Computadores

Faculdade de Ciências, Universidade do Porto

email: {nam}@ncc.up.pt

Revisão: Novembro 2000

1 Lógica Proposicional e Álgebra de Boole

A lógica proposicional remonta a Aristóteles, e teve como objectivo modelizar o raciocínio humano. Partindo de frases declarativas (*proposições*), que podem ser *verdadeiras* ou *falsas*, estuda-se o processo de construção e a veracidade de outras proposições usando conectivas como *ou* ($\vee$), *e* ($\wedge$), *não* ($\neg$), *se... então....*

Por exemplo,

Os gatos são mamíferos e *O Porto é uma cidade*

é uma conjunção de proposições verdadeiras, e portanto é uma proposição verdadeira.

George Boole (1815-1864) sistematizou a lógica proposicional como uma álgebra.

Na lógica proposicional associa-se a cada proposição um valor lógico: ou verdade (1) ou falso (0). Cada proposição é representada por uma variável (proposicional): $A, B, \ldots$.

Se A e B são proposições a expressão lógica $A \wedge B$ é uma proposição *verdadeira* se A e B o forem, e é uma proposição *falsa*, caso contrário; $A \vee B$ é uma proposição *verdadeira* se A ou B o forem, e *falsa*, caso contrário;$\neg A$ é uma proposição verdadeira se A for *falsa*, e *falsa* se A for *verdadeira*.

Formalizando temos:

Definição 1 *Dados os operadores* $\wedge$, $\vee$ *e* $\neg$ *e variáveis proposicionais* $A, B, \ldots$, *uma expressão da lógica proposicional* $E, F, \ldots$ *é definida indutivamente por:*

(i) uma variável é uma expressão lógica;

(ii) os valores lógicos 0 *ou* 1 *são expressões lógicas;*

(iii) se E *e* F *são expressões lógicas a sua conjunção* (e) *é uma expressão lógica,* $E \wedge F$*;*

(iv) se E *e* F *são expressões lógicas a sua disjunção* (ou) *é uma expressão lógica,* $E \vee F$*;*

(v) se E *é uma expressão lógica a sua negação* (não) *é uma expressão lógica,* $\bar{E}$ *ou* $\neg E$*;*

Para evitar o uso de parêntesis supõe-se que o operador $\neg$ tem maior precedência que o operador $\wedge$, e este maior precedência que operador $\vee$. Assim, a expressão $\neg\neg A \vee B$ corresponde a $(\neg(\neg A)) \vee B$ ou a $\overline{\overline{A}} \vee B$, e $\neg A \wedge B \vee C$ corresponde a $(\bar{A} \wedge B) \vee C$.

1.1 Tabelas de Verdade

Uma vez que uma variável proposicional só pode tomar o valor 1 ou o valor 0, os valores possíveis duma expressão lógica podem ser obtidos combinando os valores possíveis das variáveis que aparecem nessa expressão. Para tal constroem-se as chamadas *tabelas de verdade*, sendo apenas necessário saber inicialmente quais os possíveis valores resultantes das operações básicas: conjunção $\wedge$, disjunção $\vee$ e negação $\neg$.

As tabelas de verdade associadas aos operadores $\wedge$, $\vee$ e $\neg$ são as seguintes:

A	B	A $\wedge$ B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A $\vee$ B
0	0	0
0	1	1
1	0	1
1	1	1

A	$\neg$ A
0	1
1	0

Diz-se que duas expressões lógicas são *iguais* se assumirem os mesmos valores para uma mesma atribuição de valores às variáveis.

Sendo S um conjunto das variáveis proposicionais, o tuplo $(\{0,1\} \cup S, \vee, \wedge)$ forma uma álgebra que é designada por álgebra de Boole isto é, os operadores verificam as seguintes propriedades (axiomas):

Comutatividade

$$\begin{aligned} A \vee B &= B \vee A \\ A \wedge B &= B \wedge A \end{aligned}$$

Identidade

$$\begin{aligned} A \vee 0 &= 0 \vee A = A \\ A \wedge 1 &= 1 \wedge A = A \end{aligned}$$

Distributividade

$$\begin{aligned} A \vee (B \wedge C) &= (A \vee B) \wedge (A \vee C) \\ A \wedge (B \vee C) &= (A \wedge B) \vee (A \wedge C) \end{aligned}$$

Existência de Inversos

$$\begin{aligned} A \vee \bar{A} &= 1 \\ A \wedge \bar{A} &= 0 \end{aligned}$$

Princípio da dualidade: Das propriedades anteriores resulta que os operadores $\vee$ e $\wedge$ (e os respectivos elementos neutros 0 e 1) são duais, isto é, as propriedades válidas para um são válidas para o outro, se se substituir os elementos neutros.

Usando as propriedades anteriores podem-se ainda deduzir as seguintes propriedades (teoremas):

Associatividade

$$\begin{aligned} A \vee (B \vee C) &= (A \vee B) \vee C \\ A \wedge (B \wedge C) &= (A \wedge B) \wedge C \end{aligned}$$

Elementos absorventes

$$\begin{aligned} A \vee 1 &= 1 \\ A \wedge 0 &= 0 \end{aligned}$$

Idempotência

$$\begin{aligned} A \vee A &= A \\ A \wedge A &= A \end{aligned}$$

Complementos

$$\begin{aligned} \bar{0} &= 1 \\ \bar{1} &= 0 \\ \overline{\overline{A}} &= A \end{aligned}$$

Absorção

$$\begin{aligned} A \vee (A \wedge B) &= A \\ A \wedge (A \vee B) &= A \end{aligned}$$

Leis de DeMorgan

$$\overline{A \vee B} = \bar{A} \wedge \bar{B}$$
$$\overline{A \wedge B} = \bar{A} \vee \bar{B}$$

Para a demonstração destas propriedades podemos, alternativamente, construir tabelas de verdade. Para cada igualdade, basta verificar que para os mesmos valores das variáveis, os valores lógicos das expressões lógicas que formam cada membro da igualdade, são iguais. Por exemplo, a veracidade da lei de **DeMorgan** para a disjunção pode ser verificada pela seguinte tabela:

A	B	A $\vee$ B	$\overline{A \vee B}$	$\bar{A}$	$\bar{B}$	$\bar{A} \wedge \bar{B}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

A quarta coluna é idêntica à sétima concluindo-se assim que as expressões lógicas correspondentes são equivalentes[1].

Exercício 1.1 *Usando tabelas de verdade verifique a veracidade das restantes propriedades enunciadas anteriormente.*

Considerando a propriedade associativa dos operadores $\wedge$ e $\vee$, podemos considerar as expressões lógicas seguintes sem parêntesis:

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n$$
$$A_1 \vee A_2 \vee \ldots \vee A_n$$

Em particular podemos representar as expressões anteriores considerando apenas um operador n-ário da forma $\wedge(A_1, A_2, \ldots, A_n)$ e $\vee(A_1, A_2, \ldots, A_n)$, respectivamente.

Exercício 1.2 *Como seriam as tabelas de verdade dos operadores* $\wedge(A_1, A_2, \ldots, A_n)$ *e* $\vee(A_1, A_2, \ldots, A_n)$*?*

[1] Porque tem o mesmo valor lógico para a mesma escolha de valores das variáveis

1.2 Outros Operadores Lógicos

Habitualmente definem-se outras operações lógicas que se podem exprimir em termos das operações conjunção, disjunção ou negação. Por exemplo,

Designação	Operação	Expressão equivalente
Implicação	$A \to B$	$\neg A \vee B$
Equivalência	$A \leftrightarrow B$	$(\neg A \vee B) \wedge (A \vee \neg B)$
Ou Exclusivo	$A \dot{\vee} B$	$(A \wedge \neg B) \vee (\neg A \wedge B)$
Não-e	$A \tilde{\wedge} B$	$\neg(A \wedge B)$
Não-ou	$A \tilde{\vee} B$	$\neg(A \vee B)$

Exercício 1.3 *Escreva tabelas de verdade para cada uma das operações definidas na tabela anterior.*

Exercício 1.4 *Escreva tabelas de verdade para as expressões seguintes:*

a) $(A \to B) \leftrightarrow (\neg A \vee B)$

b) $A \to (B \to (C \vee \neg A)$

c) $(A \vee B) \to (A \wedge B)$

Dizemos que um conjunto de operadores é completo se com eles se pode exprimir as operações conjunção, disjunção e negação. Por exemplo, o operador *Não-e* (normalmente designado por `NAND`) forma um conjunto completo:

$$\begin{aligned} A \wedge B &= ((A \tilde{\wedge} B) \tilde{\wedge} 1) \\ A \vee B &= ((A \tilde{\wedge} 1) \tilde{\wedge} (B \tilde{\wedge} 1)) \\ (\neg A) &= (A \tilde{\wedge} 1) \end{aligned}$$

Exercício 1.5 *Mostre que* $\{ \tilde{\vee} \}$, $\{ \vee, \neg \}$ *e* $\{ \to, \neg \}$ *formam conjuntos completos de operadores.*

1.3 Funções Booleanas

Seja $\mathcal{BOOL} = \{0, 1\}$. Uma **função booleana** n-ária (com n argumentos) é uma função de $\mathcal{BOOL}^n$ em $\mathcal{BOOL}$. Obviamente as tabelas de verdade definem funções booleanas e, portanto, a cada expressão lógica pode-se associar uma função booleana. Em particular, os valores lógicos correspondem a funções de aridade 0, a conjunção e disjunção a funções de aridade 2, a negação a uma função de aridade 1, etc...

Dado que as funções booleanas estão definidas em conjuntos finitos, existe apenas um número finito de funções booleanas para cada aridade n. Por exemplo, as funções booleanas de aridade 1 são:

A	f(A)
0	0
1	0

A	f(A)
0	1
1	0

A	f(A)
0	0
1	1

A	f(A)
0	1
1	1

Exercício 1.6 *Existem 16 funções booleanas de aridade 2. Construa as tabelas de verdade correspondentes. E quantas funções existem de aridade n, para $n > 0$?*

Uma função booleana de aridade n pode-se representar na forma

$$f(A_1, \ldots, A_n) = E$$

onde A_i são variáveis proposicionais (ou booleanas) e E é uma expressão lógica envolvendo A_i, $1 \leq i \leq n$.

Por exemplo, $f(A, B) = A \ \wedge \ B$ é a função definida pela tabela de verdade para a conjunção. Uma tabela de verdade para uma função booleana de aridade n tem 2^n linhas, uma para cada sequência de n 0's e 1's que representam as possíveis atribuições de valores às n variáveis.

Por outro lado, dada uma função booleana definida por uma tabela de verdade podemos pretender obter uma expressão lógica a que corresponde a *mesma* tabela de verdade. Geralmente existe uma infinidade de expressões lógicas que se podem construir. No entanto, habitualmente pretendemos restringirmo-nos a um conjunto de operadores e pretendemos obter uma expressão lógica que seja, num determinado sentido, a mais "simples". A resolução deste problema é essencial para o desenho de circuitos lógicos.

A ideia básica é considerar uma expressão lógica

$$m_1 \vee m_2 \vee \ldots \vee m_k$$

onde cada m_i corresponde a uma das linhas da tabela de verdade para as quais o valor da função booleana é 1.

Denomina-se por *literal* uma expressão lógica que é uma variável A ou a negação duma variável $\bar{A}$. Uma expressão lógica está em *forma normal disjuntiva* se for a disjunção de conjunções de literais, isto é, é da forma

$$\bigvee_{i=1}^{m} \bigwedge_{j=1}^{n_i} A_{ij}$$

onde A_{ij} são literais.

Por exemplo,

$$A \vee (B \wedge D \wedge \bar{C}) \vee (\bar{B} \wedge \bar{C})$$

é uma expressão lógica em *forma normal disjuntiva.*

Usando as propriedades booleanas podemos demonstrar que qualquer expressão lógica é equivalente a uma *forma normal disjuntiva.* Analogamente, pelo princípio da dualidade, define-se *forma normal conjuntiva.*

Teorema 1 *Sendo f uma qualquer função booleana com n argumentos, $n \geq 1$, e $A_1, \ldots, A_n$ variáveis proposicional, existe uma expressão lógica E em forma normal disjuntiva tal que*

$$f(A_1, \ldots, A_n) = E$$

Dem. Se o valor de f for 0 para todos os valores dos argumentos então E é $A_1 \wedge \bar{A}_1$. Senão, para cada escolha de valores dos argumentos, que designaremos por α e que corresponde a uma linha na tabela de verdade de f, constrói-se a expressão

$$E_\alpha = A_1^\alpha \wedge A_2^\alpha \ldots \wedge A_n^\alpha$$

onde:

$$A_i^\alpha = \begin{cases} A_i & \text{se o valor de } A_i \text{ para essa escolha (linha) é 1} \\ \bar{A}_i & \text{se o valor de } A_i \text{ para essa escolha (linha) é 0} \end{cases}$$

Note que o valor lógico de E_α, para a atribuição de valores às variáveis na linha α é 1 (verifique!). Assim, para obter a expressão lógica E basta considerar a disjunção das expressões E_α tal que $f(\alpha(A_1), \ldots, \alpha(A_n)) = 1$, onde $\alpha(A_i)$ indica o valor de A_i para a escolha (linha) α. Temos

$$E = \bigvee_{f(\alpha(A_1),\ldots,\alpha(A_n))=1} E_\alpha$$

□

Considere-se a função booleana $c(A, B, C)$ definida pela seguinte tabela de verdade:

A	B	C	c(A,B,C)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Aplicando o teorema concluí-se que $c(A, B, C)$ corresponde à expressão lógica:

$$(\bar{A} \wedge B \wedge C) \vee (A \wedge \bar{B} \wedge C) \vee (A \wedge B \wedge \bar{C}) \vee (A \wedge B \wedge C)$$

Dada a analogia entre a multiplicação e adição aritméticas e a conjunção e a disjunção lógicas, ao escrever expressões lógicas substitui-se o $\wedge$ por $\cdot$ (que normalmente é omitido) e $\vee$ por $+$. No caso anterior vem:

$$\bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

Por este motivo, também se designa a forma normal disjuntiva por *soma de produtos*.

Claro, que existem outras expressões lógicas equivalentes à anterior e possivelmente mais simples. Para a construção de circuitos lógicos, interessa, obter expressões o mais simples possíveis, por causa da complexidade do circuito e do número de portas (operadores) envolvidas. Por exemplo, usando as propriedades booleanas, pode-se concluir que $c(A, B, C) = BC + AC + AB$ (verifique!).

Exercício 1.7 *Construa uma expressão lógica para cada uma das funções booleanas s, a e b definidas pelas seguintes tabelas de verdade:*

A	B	C	s(A,B,C)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

A	B	C	a	b
0	0	0	0	1
0	0	1	1	1
0	1	0	0	1
0	1	1	0	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

Tente simplificar as expressões obtidas!

1.3.1 Minterms

Um produto em que todas as variáveis aparecem exactamente uma vez, negadas ou não, designa-se por *minterm* e corresponde exactamente a uma combinação das variáveis na tabela de verdade. Assim existem exactamente, 2^n *minterm* (m_i) para n variáveis booleanas. Para duas variáveis A e B temos $\bar{A}\bar{B}$,$\bar{A}B$, $A\bar{B}$ e AB. A expressão lógica equivalente a uma tabela de verdade obtida pelo método do teorema 1 é uma soma de produtos em que todos os produtos são *minterm*, denomina-se por *soma de minterms*.

Para $n = 2$ temos

A	B	Minterm	m_0	m_1	m_2	m_3
0	0	$\bar{A}\bar{B}$	1	0	0	0
0	1	$\bar{A}B$	0	1	0	0
1	0	$A\bar{B}$	0	0	1	0
1	1	AB	0	0	0	1

Para $n = 3$ temos a seguinte tabela:

A	B	C	Minterm	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
0	0	0	$\bar{A}\bar{B}\bar{C}$	1	0	0	0	0	0	0	0
0	0	1	$\bar{A}\bar{B}C$	0	1	0	0	0	0	0	0
0	1	0	$\bar{A}B\bar{C}$	0	0	1	0	0	0	0	0
0	1	1	$\bar{A}BC$	0	0	0	1	0	0	0	0
1	0	0	$A\bar{B}\bar{C}$	0	0	0	0	1	0	0	0
1	0	1	$A\bar{B}C$	0	0	0	0	0	1	0	0
1	1	0	$AB\bar{C}$	0	0	0	0	0	0	1	0
1	1	1	ABC	0	0	0	0	0	0	0	1

Para a função, definida acima vem $c(A, B, C) = m_3 + m_5 + m_6 + m_7$.

Exercício 1.8 *Escreva a tabela de minterms para $n = 2$.*

Pelo principio da dualidade, define-se *maxterm*, *produto de somas* e *produto de maxterms*.

Exercício 1.9 *Enuncie e demonstre um teorema dual do Teorema 1 para formas normais conjuntivas.*

Exercício 1.10 *Para $f(A, B, C) = \overline{\overline{(A + B)} + \bar{A}C}$, escreva uma soma de minterms equivalente.*

1.3.2 Simplificação de funções booleanas e Mapas de Karnaugh

No caso de funções booleanas com no máximo seis variáveis existem métodos gráficos para obter uma *soma de produtos* equivalente e que é em geral mais simples do que a soma de *minterms*, obtida pelo método do Teorema 1. São os chamados *Mapas de Karnaugh* ([AU92] Capítulo 12.6, [Dew93], Capítulo 20 e [Gri99]), que são definidos em função do número de variáveis da função booleana.

Os mapas são diagramas constituídos por quadrados cada um correspondendo a um *minterm* da função booleana representada. Nos diagramas dois minterms adjacentes diferem apenas no valor de um dos literais.

Vamos supor que a expressão mais simples, como soma de produtos, é a que tem um número **mínimo** de produtos e o **menor** número de literais em cada produto. Para isso usam-se propriedades booleanas, como $\bar{A}B + AB = (\bar{A} + A)B = B$.

Duas variáveis Existem 4 minterms para uma função booleana de duas variáveis. Então o mapa é constituído 4 quadrados.

m_0	m_1
m_2	m_3

A\ B	0	1
0	$\bar{A}\bar{B}$	$\bar{A}B$
1	$A\bar{B}$	AB

Uma função é representada escrevendo um 1 nos quadrados correspondentes a cada um dos minterms da sua representação como soma de minterms.

Para as funções $f(A,B) = A + B$ e $g(A,B) = AB$ temos os mapas:

f(A,B)=A+B

A\ B	0	1
0		1
1	1	1

g(A,B)=AB

A\ B	0	1
0		
1		1

Na realidade a soma de minterms para $A+B$ é $\bar{A}B + A\bar{B} + AB$. Agora o facto de estarem 1's adjacentes na coluna do B e na linha do A permitem que a expressão seja simplificada para a forma $A + B$, isto é,

$$\bar{A}B + A\bar{B} + AB = (\bar{A}B + AB) + (A\bar{B} + AB) = (\bar{A} + A)B + A(\bar{B} + B) = B + A$$

Três variáveis Como vimos, existem 8 minterms para funções booleanas de 3 variávies. Os mapas vão ter 8 quadrados. Note que a disposição dos m_i não segue a ordem numérica, a regra é que dois quadrados adjacentes só diferem no valor de um literal.

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

$A \setminus BC$	00	01	11	10
0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$\bar{A}BC$	$\bar{A}B\bar{C}$
1	$A\bar{B}\bar{C}$	$A\bar{B}C$	ABC	$AB\bar{C}$

A representação de $f(A,B,C) = m_0 + m_2 + m_4 + m_7$ fica

$A \setminus BC$	00	01	11	10
0	1			1
1	1		1	

O 1 para ABC está isolado portanto tem de aparecer no final. Supondo que os mapas são circulares, o 1 de $\bar{A}B\bar{C}$ está a dajacente ao de $\bar{A}\bar{B}\bar{C}$ fazendo com que $\bar{A}\bar{C}(B + \bar{B})$=$\bar{A}\bar{C}$. De igual modo, os 1's na primeira coluna levam a que $\bar{C}\bar{B}(A + \bar{A}) = \bar{C}\bar{B}$, e portanto a função pode simplificar para $\bar{C}\bar{B} + \bar{A}\bar{C} + ABC$.

Para estes mapas:

- um quadrado corresponde a um minterm
- um rectângulo com dois quadrados adjacentes um produto de 2 literais
- um rectângulo com 4 quadrados adjacentes a um literal.

Claro que um mapa com tudo 1's representa 1.

Para função $c(A,B,C) = m_3 + m_5 + m_6 + m_7$ o mapa é

$A \setminus BC$	00	01	11	10
0			1	
1		1	1	1

que simplifica, por exemplo para, $c(A,B,C) = AB + AC + BC$

Para $f(A,B,C) = m_3 + m_4 + m_6 + m_7$ o mapa é

$A \setminus BC$	00	01	11	10
0			1	
1	1		1	1

que simplifica para $f(A,B,C) = BC + A\bar{C}$

Exercício 1.11 *Usando este método simplifique:*

a) $f(A,B,C) = m_0 + m_2 + m_4 + m_6$ *c)* $f(A,B,C) = m_0 + m_2 + m_4 + m_5 + m_6$

b) $f(A,B,C) = m_0 + m_1 + m_2 + m_3$ *d)* $f(A,B,C) = m_1 + m_2 + m_3 + m_6 + m_7$

Quatro variáveis Neste caso temos 16 minterms e portanto os mapas têm 16 quadrados.

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

$AD \setminus BC$	00	01	11	10
00	$\bar{A}\bar{D}\bar{B}\bar{C}$	$\bar{A}\bar{D}\bar{B}C$	$\bar{A}\bar{D}BC$	$\bar{A}\bar{D}B\bar{C}$
01	$\bar{A}D\bar{B}\bar{C}$	$\bar{A}D\bar{B}C$	$\bar{A}DBC$	$\bar{A}DB\bar{C}$
11	$AD\bar{B}\bar{C}$	$AD\bar{B}C$	$ADBC$	$ADB\bar{C}$
10	$A\bar{D}\bar{B}\bar{C}$	$A\bar{D}\bar{B}C$	$A\bar{D}BC$	$A\bar{D}B\bar{C}$

Para a função $f(A,B,C,D) = m_0 + m_2 + m_1 + m_3 + m_8 + m_9 + m_{10}$,

$AD \setminus BC$	00	01	11	10
00	1	1	1	1
01				
11				
10	1	1		1

Note que supõe que os diagramas são toros ("donuts") em que as extremidades superior e inferior e esquerda e direita estão ligadas.

Combinando os 1's nos quatro cantos temos $\bar{A}\bar{D}\bar{B}\bar{C} + \bar{A}\bar{D}B\bar{C} + A\bar{D}\bar{B}\bar{C} + A\bar{D}B\bar{C} = \bar{D}\bar{C}$ Analogamente, os da primeira linha dão $\bar{A}\bar{D}$. Finalmente os dois primeiros 1's da última linha podem combinar com os dois prineiros da primeira linha

$$\bar{A}\bar{D}\bar{B}\bar{C} + \bar{A}\bar{D}\bar{B}C + A\bar{D}\bar{B}\bar{C} + A\bar{D}\bar{B}C = \bar{D}\bar{B}$$

vem $f(A, B, C, D) = \bar{D}\bar{C} + \bar{A}\bar{D} + \bar{D}\bar{B}$

As combinações de quadrados que se podem escolher para $n = 4$ são:

- um quadrado representa um minterm de 4 literais
- um rectângulo com 2 quadrados adjacentes representa um produto de 3 literais
- um rectângulo com 4 quadrados adjacentes representa um produto de 2 literais
- um rectângulo com 8 quadrados adjacentes representa 1 literal

Resumindo, para simplificar uma função booleana dada pelo seu mapa de Karnaugh:

1. Começar por combinar os termos na tabela para os quais existe apenas uma possibilidade de simplificação.
2. Verificar os quatro cantos do mapa, que podem conter 1's adjacentes
3. Tentar encontrar o maior número de 1's adjacentes
4. Se houver uma escolha no modo de simplificar, deve-se usar 1's que ainda não tenham sido usados para simplificar

Exercício 1.12 *Simplifique*

a) $f(A,B,C,D) = m_9 + m_{10} + m_{11} + m_{12} + m_{13}$

b) $f(A,B,C,D) = m_3 + m_4 + +m_5 + m_7 + m_9 + m_{13} + m_{14} + +m_{15}$

Este método torna-se impraticável para funções com mais do que 6 variávies. O método Quine-MaCluskey resolve o problema de minimização de somas de produtos para qualquer número de literais.

2 Circuitos Lógicos

A lógica proposicional pode ser usada para a implementação de circuitos electrónicos, que estão na base da construção das principais componentes dum computador electrónico digital. Nesses circuitos usam-se dois níveis de voltagem para representar os valores binários 0 e 1.

Os circuitos são constituídos por *portas* que admitem uma ou várias entradas, cada uma podendo assumir o valor 0 ou 1. Usualmente, têm uma saída que é função das entradas, que pode ser também 0 ou 1. As portas usadas nos circuitos dependem da tecnologia (por exemplo com transístores de algum tipo), mas correspondem normalmente às operações lógicas $\wedge$ (`e`), $\vee$ (`ou`) e $\neg$ (`nao`, inversor); ou $\tilde{\wedge}$ (`nao-e`) ou $\tilde{\vee}$ (`nao-ou`). As portas são combinadas em *circuitos* ligando-se (electricamente) as saídas de algumas portas às entradas de outras. Um circuito como um todo pode ter várias entradas e várias saídas.

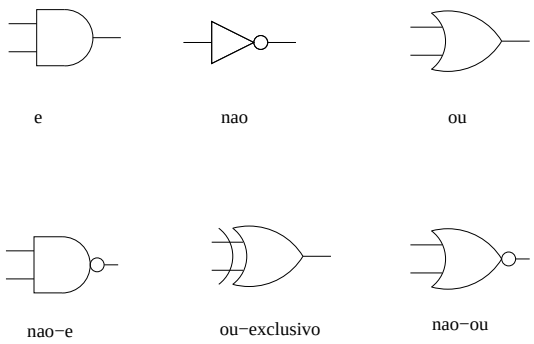

Figura 1: Símbolos para as portas

Dada uma expressão lógica existe sempre um circuito lógico que corresponda à mesma

função booleana.[2]

Nestes circuitos, o valor de saída num dado instante depende exclusivamente dos valores de entradas. No entanto, existem circuitos lógicos que não correspondem directamente a funções booleanas porque a saída duma porta pode estar ligada, através de outras portas, a uma entrada dessa porta, como ilustra a Figura 2.

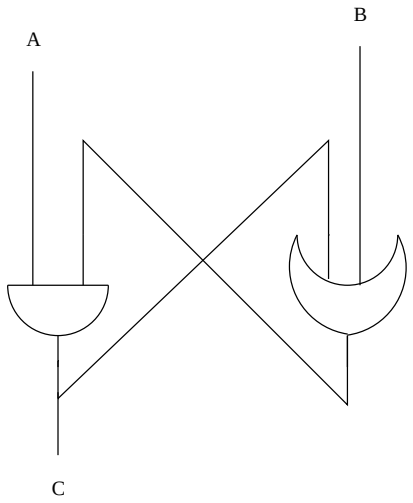

Figura 2: Circuito sequencial

Os primeiros são denominados *circuitos combinatórios* e os segundos *sequenciais*. Nos circuitos combinatórios, os valores das saídas num dado instante dependem apenas do valor das entradas num instante imediatamente anterior. Nos circuitos *sequenciais* existem, para além de portas, "unidades de memória" de modo a que os valores das saídas num dado instante dependem também do estado das unidades de memória. O comportamento de um circuito sequencial tem de ser especificado em função do tempo para os possíveis valores das entradas e dos estados internos das memórias.[3] No circuito da Figura 2, suponhamos que A e B tem o valor 1. Então a saída da porta `ou` é 1, e assim, as entradas na porta `e` são 1 e a sua saída é 1. Se B tiver o valor 0, o valor de C não se altera. No entanto, se A for (tornar) zero o valor de C também passa a ser 0, independentemente do valor de B.

A capacidade "recordar" dos circuitos sequenciais permite-os serem usados na construção das unidades de memória e dos registos.

[2]Ver [AU92] Capítulo 13, [Dew93] 3, 13 e 28 ou um livro sobre desenho de circuitos, como, [MK97] Cap 2.8-2.9.

[3]Em termos formais os circuitos sequenciais correspondem a autómatos finitos determinísticos.

Por outro lado, os circuitos combinatórios são usados para implementar operações aritméticas, descodificadores, selectores e outras componentes de controlo.

Nas secções seguintes iremos dar alguns exemplos de circuitos combinatórios e depois referir como se pode implementar uma unidade de memória.

2.1 Somadores

O somador de dois bits (*half-adder*) é um circuito combinatório que tem duas entradas, os bits B_0 e B_1, e produz duas saídas: o bit soma S e o bit de transporte T. A tabela de verdade da função booleana correspondente é

B_1	B_0	S	T
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

As expressões lógicas para a soma e o transporte podem ser

$$\begin{aligned} \mathrm{S} &= \bar{B}_1 B_0 + B_1 \bar{B}_0 = B_1 \dot{\vee} B_0 = (\bar{B}_1 + \bar{B}_0)(B_1 + B_0) = B_0 \dot{\vee} B_1 \\ T &= B_1 B_0 \end{aligned}$$

Um circuito lógico para a implementação do somador de dois bits, é apresentado na Figura 3.

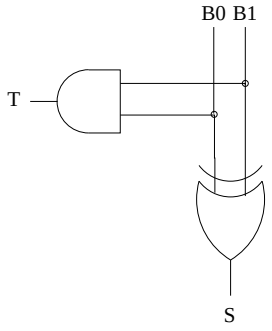

Figura 3: Somador de 2 bits

Exercício 2.1 *Usando outras expressões lógicas, desenhe outro circuito lógico para a implementação do somador de dois bits.*

Para se obter um somador de números em binário é necessário considerar um somador de dois bits e um transporte (*full-adder*) que produza um bit soma e um transporte. As respectivas tabelas de verdade são:

B_1	B_0	T_0	S	T
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A tabela de verdade para o transporte T coincide com a tabela apresentada na página 10,e portanto, como vimos, podemos obter uma expressão lógica (como soma de produtos) mais simples para T:

$$\begin{aligned} \mathrm{S} &= \bar{B}_1\bar{B}_0T + \bar{B}_1B_0\bar{T} + B_1\bar{B}_0\bar{T} + B_1B_0T \quad = \quad B_1\dot{\vee}B_0\dot{\vee}T \\ \mathrm{T} &= B_0T_0 + B_0B_1 + B_1T_0 \\ &= B_0B_1 + T_0(B_1\bar{B}_0 + B_1\bar{B}_1) \\ &= B_0B_1 + T_0(B_0\dot{\vee}B_1) \end{aligned}$$

O somador de números não negativos binários com n bits pode ser construído com um somador de dois bits e $n-1$ somadores de dois bits e transporte. Por exemplo, para $n = 3$ temos

$$\begin{array}{cccc} & B_2 & B_1 & B_0 \\ + & B_2' & B_1' & B_0' \\ \hline T_2 & S_2 & S_1 & S_0 \end{array}$$

e o esquema do circuito lógico é apresentado na Figura 5.

Exercício 2.2 *Descreva uma expressão lógica e um circuito lógico que calcule a diferença de dois valores de 1 bit $D1-D2$. No caso de $D1 = 0$ e $D2 = 1$, a diferença é 1 activando-se a linha de erro E.*

Para cada um dos exercícios seguintes tente simplicar a expressão lógica obtida, antes do desenho do circuito.

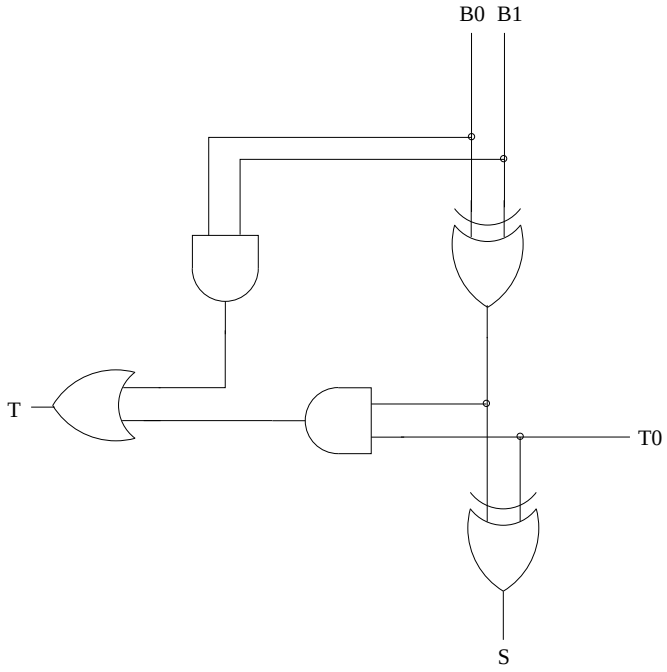

Figura 4: Somador de 3 bits

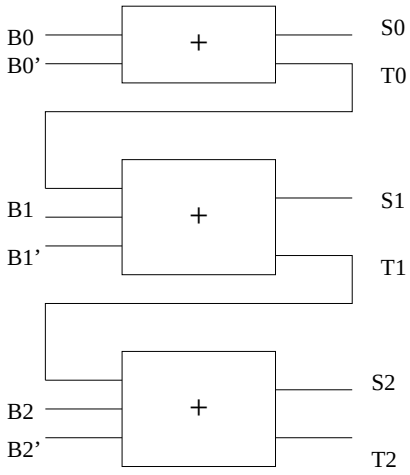

Figura 5: Somador para números de 3 bits

Exercício 2.3 *Descreva uma expressão lógica e um circuito lógico que dado um inteiro V com 3 bits determine o seu complemento para 2. Considere uma linha que assinale se houve erro. Generalize para inteiros de n bits. Sugestão: construa um circuito lógico que inverta os bits (transforme 0 em 1 e 1 em 0) dum inteiro dado (***Inversor***) e um circuito lógico que adicione 1 a um valor dado (*Incrementador*).*

Exercício 2.4 (Deslocação de 1 bit)

(i) Descreva um circuito lógico que dado um valor binário de 3 bits (linhas de dados) $b_2b_1b_0$ produza um resultado $s_2s_1s_0$ e mais um bit e (quatro linhas de saída) tal que $s_0 = 0$, $s_1 = b_0$, $s_2 = b_1$ e $e = b_2$, isto é, os "bits" são deslocados uma posição para a esquerda. Conclua que esta operação corresponde a uma multiplicação de um número positivo em binário por 2. Se $e = 1$ isso significa que ocorreu um erro, isto é, não é possível representar o resultado da multiplicação com 3 bits.

(ii) Considere o problema idêntico ao da alínea anterior mas com uma deslocação de um bit para a direita. Que operação aritmética é implementada?

(iii) (*) Descreva como pode ser implementado um circuito para multiplicar dois inteiros não negativos em binário com n-bits. Suponha $n = 3$.[4]

(iv) (*) Descreva como pode ser implementado um circuito para dividir dois inteiros não negativos em binário com n-bits. Suponha $n = 3$.

Exercício 2.5 *Descreva um circuito lógico que dado um valor em binário com 3 bits $b_2b_1b_0$ e outro valor em binário com dois bits d_1d_0, calcule o valor $s_2s_1s_0$ que resulta do primeiro deslocando para a esquerda o número de bits indicado pelo segundo, por iteração da operação descrita no Exercício 2.4, (i). Que operação aritmética é implementada?*

2.2 Restrições físicas dos circuitos

Actualmente os circuitos lógicos são construídos como *chips*, ou circuitos integrados. As tecnologias mais usadas actualmente para este tipo de circuitos são as CMOS (*complementary metal-oxide semiconductor*). Um *chip* é constituído por camadas de materiais, que combinadas, formam as *portas* e as conexões entre elas. A espessura duma linha de ligação

[4] Pode consultar [PH94].

pode ser menor do que um *mícron* e uma porta pode ser construída em alguns mícrons. Um chip pode ter cerca de um $1cm^2$ e conter milhões de portas. Para a construção de cada porta utilizam-se transístores que são *interruptores* ligado/desligado controlados electricamente. Para uma descrição mais aprofundada ver por exemplo [PH94], Capítulo 1 e [Dew93], Capítulos 56. Cada tipo de tecnologia utilizada impõe restrições no desenho dos circuitos lógicos. Vamos referir algumas das mais importantes.

Atraso Associada a cada porta existe um *atraso*, entre o tempo que as linhas de entrada ficam activas e o tempo em que a saída fica disponível. Este atraso pode ser de alguns nano-segundos (10^{-9}) mas, num circuito complexo[5], como a informação se propaga por muitos níveis de portas esse factor é não negligenciável. Dum modo geral, o número de portas através do qual um valor tem ser propagado, deve ser minimizado. O *atraso* de um circuito é definido como o número máximo de portas que é necessário atravessar entre alguma entrada e alguma saída do circuito. Por exemplo para o somador de dois bits (com ou sem transporte),com portas `e`, `ou` e `não` o atraso é 3.

Propagação do atraso Para além do número de portas, a área ocupada pelo circuito é também um factor essencial. Se a área é grande, entre as portas são necessárias linhas (condutores eléctricos) mais compridas, e portanto mais tempo demora o sinal a chegar duma porta a outra.

Fan in* e *Fan out Para uma porta, existem limitações físicas ao seu número de entradas (*fan in*) e ao número de entradas a que uma saída sua está ligada (*fan out*). Na prática, quantas mais *fan in/fan out*, mais lenta é a porta.

Em conclusão, o desenho de circuitos lógicos é um factor essencial na construção dos computadores actuais e para o seu desempenho eficiente. Para a construção dos circuitos, não só se utilizam resultados da lógica proposicional para obter expressões lógicas e as minimizar, as outras técnicas da programação são utilizadas para minimizar os custos físicos da sua construção.

Por exemplo, o somador de dois números inteiros com n-bits apresentado na secção anterior (para $n = 3$), tem um atraso de $3n$. No entanto, utilizando técnicas do desenho de algoritmos é possível construir um somador com um atraso apenas[6] de 3 [AU92], Cap. 13.6.

[5] Em computadores que executam instruções em $\leq 10^{-6}$ segundos.

[6] Deste ponto de vista, tão rápido como um somador de dois bits.

2.3 Comparadores

Um *comparador* é um circuito combinatório que permite comparar o valor absoluto de dois inteiros A e B representados em binário com n-bits. Tem três saídas que indicam respectivamente se $A < B$, $A = B$ ou $A > B$. Vamos considerar apenas o caso de inteiros não negativos. Se $n = 1$ a tabela de verdade associada é

A_0	B_0	O_1	O_2	O_3
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

$$\begin{array}{rcll} & O_1 & = & \bar{A}_0 B_0 & (A < B) \\ \text{e} & O_2 & = & \bar{A}_0 \bar{B}_0 + A_0 B_0 & (A = B) \\ & O_3 & = & A_0 \bar{B}_0 & (A > B) \end{array}$$

Exercício 2.6 *Desenhe um circuito para o comparador de dois números de 1 bit.*

Exercício 2.7 *Calcule a tabela de verdade e desenhe um circuito lógico para um comparador de dois números de dois bits A_1A_0 e B_1B_0. Nota: a tabela tem 16 linhas! Tente simplificar as expressões lógicas obtidas.*

2.4 Descodificadores

Um *descodificador* é um circuito combinatório que, geralmente, traduz n bits de entrada em $m \leq 2^n$ bits de saída, onde apenas uma das saídas é 1 para cada uma das possíveis valores dos entradas. Permite traduzir uma entrada de n-bits numa saída que corresponde ao valor que a sequência de bits de entrada representa em binário. Por exemplo um descodificador 3 para 8, tem 3 linhas de entrada e 8 linhas de saída e a sua tabela de verdade é:

B_2	B_1	B_0	L_7	L_6	L_5	L_4	L_3	L_2	L_1	L_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Exercício 2.8 *Desenhe um circuito lógico para um descodificador* 3 *em* 8.

Exercício 2.9 *Calcule a tabela de verdade e desenhe um um circuito lógico para um descodificador* 2 *em* 4.

Um *codificador* é um circuito lógico inverso do anterior. Dadas 2^n entradas produz uma saída de n bits. A tabela de verdade par um codificador 4 em 2 é:

L_3	L_2	L_1	L_0	B_1	B_2
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Exercício 2.10 *Escreva uma expressão lógica e um circuito para o codificador 4 em 2. Construa uma tabela de verdade e um circuito lógico para um codificador* 8 *em* 3.

2.5 Multiplexadores

Um *multiplexador* é um circuito combinatório que selecciona uma entre 2^n entradas A_0, ..., A_{2^n-1} e liga-a a uma saída comum. Para seleccionar uma entrada A_i possuí n entradas de controlo C_1, ... C_n, que são interpretadas como a representação binária do inteiro i. Os multiplexadores são muito usados nos computadores pois permitem que diferentes unidades utilizem um mesmo dispositivo (ligado à linha de saída única). Por exemplo, um mesmo controlador pode controlar vários discos.

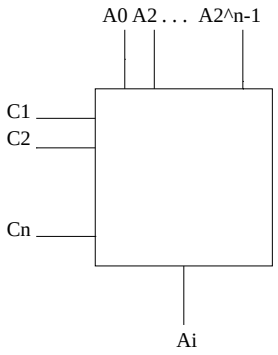

Figura 6: Esquema de um multiplexador

Esquematicamente um multiplexador de n linhas é apresentado na Figura 6.

Um 1-multiplexador permite seleccionar um de dois bits. A expressão lógica associada à saída é $A_i = A_0\bar{C}_1 + A_1C_1$. Se C_1 for 1 é $A_i = A_1$, se for 0 $A_i = A_0$.

Um 2-multiplexador permite seleccionar um de 4 bits. A expressão lógica associada à saída é

$$A_i = A_0\bar{C}_2\bar{C}_1 + A_1\bar{C}_2C_1 + A_2C_2\bar{C}_1 + A_3C_2C_1$$

Exercício 2.11 *Desenhe um circuito lógico para um 1-multiplexador e para um 2-multiplexador.*

Exercício 2.12 *Obtenha uma expressão lógica para um 3-multiplexador.*

Exercício 2.13 *Num mostrador digital, cada algarismo (decimal) é formado por 7 segmentos. A activação, ou não, de cada um dos segmentos determina o algarismo representado. Descreva um circuito lógico que dados 3 bits (em linhas de dados) da representação em binário de um algarismo (inferior a 8), $b_2b_1b_0$, actue sobre cada um dos segmentos de acordo com esse valor.*

2.6 Elementos de memória e Manipulação de informação

Uma *memória* de um bit é um circuito capaz de *recordar* ao longo do tempo o valor da sua entrada e produzir esse valor.

Um destes circuitos muito simples é apresentado na Figura 7. O circuito é controlado pela linha de entrada L, que permite a actualização do valor "guardado" pelo circuito. Normalmente, o valor dessa linha é 0 e o valor da saída permanece, ao longo do tempo, sempre o mesmo. Efectivamente, se $L = 0$ e $O = 1$ então, as entradas na porta b são ambas 1 e portanto a saída da porta d é 1, isto é, se o valor de O é 1 permanece 1. Por outro lado, se o valor de $L = 0$ e $O = 0$, então uma das entradas da porta b é 0, o que significa que a sua saída é também 0. Assim as duas entradas da porta d são 0, produzindo um valor 0 para O. Suponhamos que o valor de L passa a 1. Então, por causa do inversor a a saída da porta b é 0. Por outro lado, a primeira entrada da porta c é 1, portando o valor da saída desta porta é o valor de I e que será também o valor de O. Isto é, se $L = 1$ o valor de saída do circuito é valor da entrada I. Se L passar para 0, o valor de saída do circuito permanecerá esse valor.

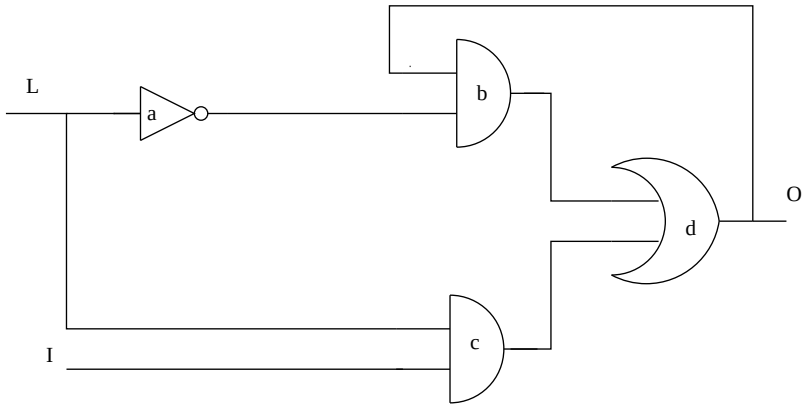

Figura 7: Uma memória simples de 1 bit

Na prática, os elementos de memória, denominados *flip-flops*, são mais complexos e, em particular, tem associados um circuito de relógio que permite controlar temporalmente (sincronizar) as várias componentes dos circuitos sequenciais. Os circuitos de relógio são elementos que apenas emitem um sinal com uma frequência bem determinada.

Na Figura 8 está representado um *flip-flop* constuído com portas *nao-e*. Fácilmente se vê que as saídas `Q` e `Q'`, não podem ser simultaneamente `0`. Porque, as portas *nao-e* têm saída `1` se uma das entradas é `0`. Assim, temos também que supor que as entradas `R` e `S` não são simultaneamente `0`.

Podemos então considerar apenas dois estados:

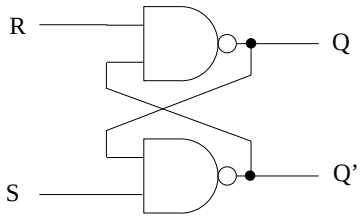

Figura 8: Um flip-flop

Estado	Q	Q'
0	0	1
1	1	0
ind.	1	1

No estado 1 guarda (em Q) um 1, no estado 0 guarda um 0 Assim consoante o estado (0 ou 1) e os valores de R e S, o *flip-flop* muda de estado, segundo a seguinte tabela:

Estado Inicial	R	S	Q	Q'	Estado Final
1	1	0	0	1	0
1	1	1	1	0	1
1	0	1	1	0	1
0	0	1	1	0	1
0	1	1	0	1	0
0	1	0	0	1	0

Exercício 2.14 *Verifique a tabela anterior está correcta.*

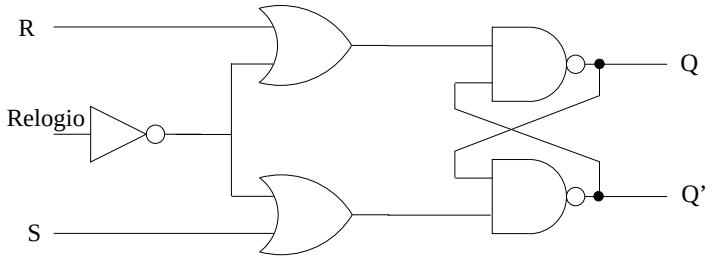

Figura 9: Um flip-flop com relógio

A introdução de um relógio no circuito, ver Figura 9 permite que apenas quando o relógio pulse (1), o valor de S ou R "passe" para o *flip-flip* e eventualmente mude o seu estado. Se

`S` é `1`, o *flip-flop* passa a guardar um `1` (se já não estava nesse estado). Se `R` é `1`, o *flip-flop* passa a guardar um `0` (se já não estava nesse estado).

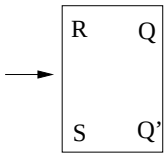

Figura 10: Esquema de um *flip-flop*

Podemos representar o circuito anterior apenas como uma caixa com 3 entradas e duas saídas (Figura 10).

Registos e Memória Principal Um **registo** de n-bits é um conjunto de n memórias de 1 bit. Normalmente, os registos são de 8, 16 ou 32 bits. Num CPU (unidade central de processamento) existem habitualmente vários registos (fixos) para tarefas específicas ou para tarefas gerais (*registos* programáveis). Num registo, são efectuadas as operações de leitura ou de escrita simultaneamente em cada um dos seus bits.

A **memória principal** é constituída por posições memória todas com o mesmo número de bits, normalmente 8 (1 byte), tendo cada uma delas um **endereço** (valor numérico). Num computador a capacidade memória (número de posições de memória) pode variar mas está normalmente condicionado pelo número de endereços que o seu CPU pode aceder.

Para exemplificar o acesso de leitura/escrita a uma posição de memória, vamos considerar um módulo de memória com 4 posições de memória com 3 bits cada uma. A cada posição de memória associa-se um endereço que é um inteiro de 0 a 3. Serão necessários 2 bits para representar um endereço, no caso geral, para 2^n posições de memória serão necessários n bits. Supomos que existe um *registo de endereço* (RE) que contém o endereço duma posição de memória a que se quer aceder e um *registo de depósito de conteúdo* (RC) que contém um valor a escrever para a memória ou que irá conter um valor lido da memória.

Na Figura 11 mostra-se um esquema do circuito de escrita. O registo RE tem 2 bits, o registo RC tem 3 bits e cada uma das posições de memória, à esquerda, também tem 3 bits. Cada bit é um elemento de memória, como o exemplificado acima. Habitualmente, uma memória de um bit é representada por um quadrado que pode conter o valor 0 ou 1, consoante o valor da sua linha de saída O. A esse valor chama-se também o seu *conteúdo*.

Na figura, supõe-se que o registo RE contêm num dos elementos o valor 0 e no outro o valor 1. A sequência de bits 01 corresponde em binário ao endereço da posição de memória que se irá manipular.

Para uma operação de escrita, supomos que existe uma linha W tal que só se o seu valor for 1 é que a alteração duma posição de memória é permitida.

O registo de endereço está ligado a um descodificador que activa uma das suas 2^n saídas de acordo com o valor da sequência de n bits de entrada. Neste caso, como $n = 2$, o descodificador activa uma das suas 4 saídas, cada uma das quais está ligada a uma das posições de memória. Assim é identificada qual a posição de memória que se pretende alterar.

O registo de depósito de conteúdo RC, isto é, cada um dos seus bits, está ligado com cada uma linhas de entrada I de cada posição de memória (isto é, com cada um dos respectivos bits). No esquema, a linha com uma seta na extremidade – que liga o RC a cada posição de memória – , corresponde na realidade a 3 linhas uma para cada um dos bits das posições de memória.

Quando o valor de W é 1 então, considerando os circuitos dos elementos de memória e o descodificador, facilmente se conclui que cada um dos bits da posição de memória cujo endereço é conteúdo de RE vai ficar com os valores dos respectivos bits do RC. Neste caso, a posição de memória 1 ficaria com a sequência 110.

A leitura de um valor em memória é uma tarefa mais simples, uma vez que não se tem de alterar o estado dos elementos de memória. Apenas, é necessário fornecer o endereço da posição de memória, como entrada, e a saída, será a informação contida nessa posição de memória (o seu conteúdo).

Na Figura 12 apresenta-se um esquema desse circuito. Para além do registo de endereço, do registo de depósito de conteúdo e das posições de memória, é necessário um multiplexador. Este multiplexador tem uma linha de entrada por cada bit de cada uma das posições de memória ($3 \times 4 = 12$) e duas linhas de entrada de controlo. Consoante o valor correspondente às linhas de controlo são seleccionadas as 3 linhas da posição de memória cujo endereço é esse valor. Assim o conteúdo dessa posição de memória é "lido" e guardado no registo de depósito RC. Como no esquema anterior, as linha com uma seta na extremidade correspondem a 3 linhas, uma para cada bit das posições de memória. Neste caso, a posição de memória seleccionada foi a 3 (supondo numeradas de cima para baixo) e portanto a sequência 011 foi lida para o registo RC.

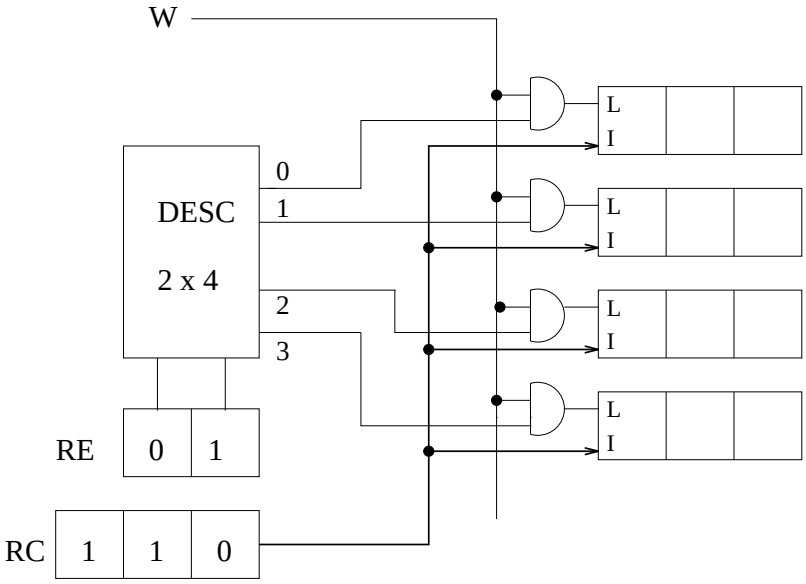

Figura 11: Escrita numa memória 4×3

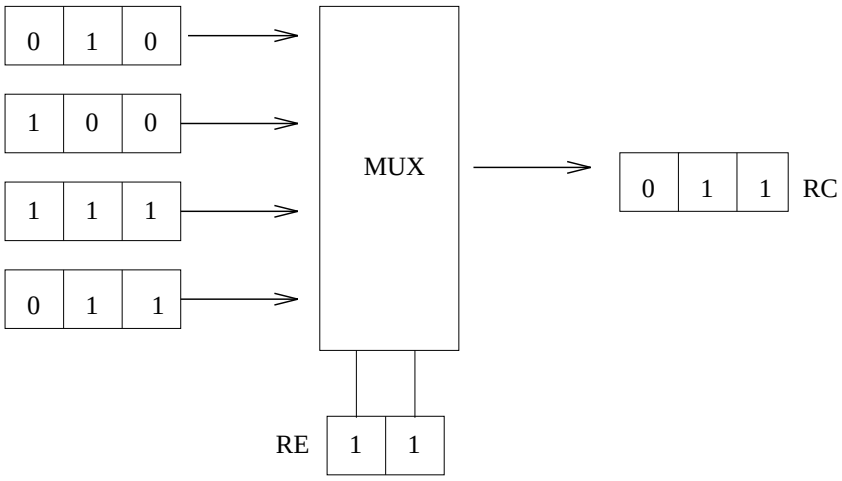

Figura 12: Leitura numa memória 4×3

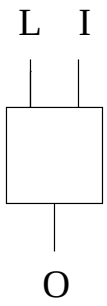

Figura 13: Esquema duma memória de um bit

Exercício 2.15 *Representando uma memória de um bit pelo esquema da Figura 13, indique, fazendo um esquema:*

a) como podem ser ligadas cada uma das memórias de um bit duma mesma posição de memória (ou registo) para serem simultaneamente escritas. Amplie o esquema da Figura 12 considerando as 3 linhas de entrada, uma para cada bit das posições de memória e registo RC.

b) como podem ser ligadas cada uma das memórias de um bit duma mesma posição de memória (ou registo) para serem simultaneamente lidas. Amplie o esquema da Figura 12 considerando as 3 linhas de saída de cada posição de memória.

Exercício 2.16 *Desenhe o circuito do multiplexador usado na Figura 12.*

Exercício 2.17 *Faça um esquema que englobe simultaneamente os circuitos de leitura e escrita numa memória* 4×3*, apresentados nas Figuras 11 e 12.*

Exercício 2.18 *Leia o texto* Redes Neuronais e Computadores Digitais*, [Fil92], para que, dum modo simples, aprenda mais sobre a estrutura e funcionamento das várias componentes dum computador digital.*

Referências

[AU92] Alfred V. Aho and Jeffrey D. Ullman. *Foundations of Computer Science.* Computer Science Press, W. H. Freeman and Company, 1992. Capítulos: 12;13.

[Dew93] A. K. Dewdney. *The (New) Turing Omnibus – 66 Excursions in Computer Science*. W. H. Freeman and Company, 1993.

[Fil92] Miguel Filgueiras. Redes neuronais e computadores digitais. Technical report, CIUP, 1992.

[Gri99] Ralpf P. Grimaldi. *Discrete and Combinatorial Mathematics*. Addison Wesley, 1999. Capítulos: 15.1 e 15.2.

[MK97] M. Morris Mano and Charles R. Kime. *Logic and Computer Design Fundamentals*. Prentice-Hall, 1997.

[PH94] David A. Patterson and John L. Hennessy. *Computer Organization & Design: The Hardware / Software Interface*. Morgan Kaufmann Publishers, 1994.